

JavaScript

Igor Wojnicki

Katedra Informatyki Stosowanej
Akademia Górniczo-Hutnicza w Krakowie

8 stycznia 2014

\$Id: 04javascript.tex,v 1.1 2014/01/08 08:48:49 wojnicki Ex

Spis Treści I

- 1 Wstęp
- 2 Elementy Języka
- 3 Sterowanie
- 4 Funkcje
- 5 Obiekty
- 6 HTML
- 7 Przeglądarka

I. Wojnicki, JiTW

Spis Treści

- 1 Wstęp
- 2 Elementy Języka
- 3 Sterowanie
- 4 Funkcje
- 5 Obiekty
- 6 HTML
- 7 Przeglądarka

I. Wojnicki, JiTW

Typowe zastosowania

- mechanizm “Cookie”,
- manipulacja przeglądarką (okna, panele),
- walidacja formularzy,
- manipulacja dokumentem (DOM),
- (nie)proste aplikacje.

Wersje

JavaScript vs. JScript vs. ECMAScript (European Computer Manufacturers Association)

wersja	przeglądarka	standard
1.0	Navigator 2.0, IE 3.0 (wczesne)	ECMA Script edition 1/2
1.1	Navigator 3.0, IE 3.0 (późniejsze)	
1.2	Navigator 4-4.05	
1.3	Navigator 4.06-4.5, IE 4.0	
1.4	Netscape Server	ECMA Script edition 3
1.5	Navigator 6.0/Mozilla, IE 5.5, 6.0	
1.6	Gecko 1.8 (Firefox 1.5)	
1.7	Gecko 1.8 (Firefox 2)	
1.8	Gecko 1.9 (Firefox 3)	

Dokumentacja

- <http://developer.mozilla.org/en/docs/JavaScript>
- <http://www.ecma-international.org/publications/standards/Ecma-262.htm>
- **EchoEcho.com**
<http://www.echoecho.com/javascript.htm>.

Spis Treści

- 1 Wstęp
- 2 Elementy Języka**
- 3 Sterowanie
- 4 Funkcje
- 5 Obiekty
- 6 HTML
- 7 Przeglądarka

I. Wojnicki, JiTW

Komentarze

- `//` komentarz pojedynczej linii
- `/*` komentarz
wielo
liniowy dowolnej
długości
`*/`

Typy danych

- liczba np. `3.1415`, `-3.1e5`, `42`, `0x10`, `05`,
- wartość logiczna: `true` albo `false`,
- łańcuch znaków,
np. `"Dzień Dobry"`, `'Do widzenia'`,
- `null`, tzw. "nic",
- `undefined`, wartość niezdefiniowana.

Uwaga:

- JavaScript rozróżnia duże i małe litery.
- Słaba (dynamiczna) kontrola typów danych (weakly typed).
- Konwersja typów automatyczna.

Znaki specjalne

znak	znaczenie
<code>\b</code>	Backspace
<code>\f</code>	Form feed
<code>\n</code>	New line
<code>\r</code>	Carriage return
<code>\t</code>	Tab
<code>'</code>	Apostrof
<code>"</code>	Cudzysłów
<code>\\</code>	Backslash

Znaki specjalne cd.

znak	znaczenie
<code>\XXX</code>	znak o kodzie XXX (ósemkowo)
<code>\xXX</code>	znak o kodzie XX (szesnastkowo)
<code>\uXXXX</code>	znak o kodzie XXXX wg. UTF-16/UCS-2

Operatory porównujące

operator	znaczenie
==	równe
!=	różne
===	równe, ten sam typ
!==	różne, ten sam typ
>	większe
>=	większe lub równe
<	mniejsze
<=	mniejsze lub równe

Operatory arytmetyczne

operator	znaczenie
+ - * /	
%	Modulo (reszta z dzielenia)
++	inkrementacja
--	dekrementacja
-	negacja

Operatory logiczne

operator	znaczenie
<code>ex1 && ex2</code>	logiczne 'i'; jest równe <code>ex1</code> gdy <code>false</code> , jest równe <code>ex2</code> gdy <code>true</code>
<code>ex1 ex2</code>	logiczne 'lub'; jest równe <code>ex1</code> gdy <code>true</code> , jest równe <code>ex2</code> gdy <code>false</code>
<code>!</code>	przeczenie

Operatory bitowe

operator	znaczenie
&	AND
	OR
^	XOR
~	negacja
<<	w lewo
>>	w prawo

I. Wojnicki, JiTW

Operatory specjalne

- `+` – łączenie łańcuchów znaków,
- `delete` – usuwanie obiektu, własności obiektu, bądź elementu tablicy,
- `new` – tworzenie obiektu,
- `this` – słowo kluczowe umożliwiające odwołanie się do obiektu wywołującego metodę,
- `typeof` – typ obiektu (`boolean`, `string`, `number`, `object`, `undefined` (dla skasowanego przy użyciu `delete`))
- `void` – oblicza wyrażenie z zaniedbaniem wartości,
- wsparcie dla wyrażeń regularnych.

Spis Treści

- 1 Wstęp
- 2 Elementy Języka
- 3 Sterowanie**
- 4 Funkcje
- 5 Obiekty
- 6 HTML
- 7 Przeglądarka

I. Wojnicki, JiTW

Instrukcje sterujące

- `if (warunek) {`
 `...`
}
- `else {`
 `...`
}
- `switch(wyrażenie) {`
 `case wartość1 :`
 `...`
 `break;`
 `case wartość2 :`
 `...`
 `break;`
 `...`
 `default :`
 `...`
}

Pętle

- `for (wyr_pocz; warunek; wyr_inkrement) {`
 `...`
 `}`
- `do {`
 `...`
 `} while(wyrazenie)`
- `while (wyrazenie) {`
 `...`
 `}`

Kontrola pętli

- `break`
przerywa działanie pętli,
- `continue`
przerwanie bieżącego i wykonanie następnego kroku pętli.

I. Wojnicki, JiTW

Operacje na obiektach

- `for (zmienna in obiekt-albo-tablica) {`
 ...
}
- `with (obiekt){`
 ...
}

I. Wojnicki, JTW

Spis Treści

- 1 Wstęp
- 2 Elementy Języka
- 3 Sterowanie
- 4 Funkcje**
- 5 Obiekty
- 6 HTML
- 7 Przeglądarka

I. Wojnicki, JiTW

Funkcje

- definicja:

```
function nazwa(arg1, arg2, ... argn) {  
    return wartość;  
}
```

Argumenty funkcji przekazywane są niejawnie z wykorzystaniem obiektu `arguments`: `arguments[3]`, `arguments.length`
Ilość argumentów nie jest sprawdzana.

- wywołanie:

```
zmienna=mojaFunkcja(argument)
```

- argumenty przekazywane przez *wartość*, z wyjątkiem obiektów, które są przekazywane jako *referencje*.

Predefiniowane funkcje

- `eval` (łańcuch)
- `parseInt` (łańcuch)
- `parseFloat` (łańcuch)
- `isNaN` (wartość)

I. Wojnicki, JiTW

Spis Treści

- 1 Wstęp
- 2 Elementy Języka
- 3 Sterowanie
- 4 Funkcje
- 5 Obiekty**
- 6 HTML
- 7 Przeglądarka

I. Wojnicki, JiTW

Obiekty - wstęp

- *Obiekt* jest abstrakcyjnym konstruktem posiadającym własności, które są obiektami bądź zmiennymi.
- Odwołanie się do własności obiektu:
`nazwaObiektu.nazwaWłasności`
albo
`nazwaObiektu["nazwaWłasności"]`

Należy zachować konwencje odwoływania się do własności obiektu!

Predefiniowane typy obiektów

- Array
- Boolean
- Date
- Math
- Number
- String
- Function – do dynamicznego tworzenia funkcji:
`new Function(argumenty, ciało)`

Obiekt typu Array

- tworzenie obiektu

```
mojaTablica0 = new Array(ileElementów);  
mojaTablica1 = new Array(e1, e2 ... en);  
mojaTablica  = ["1element", "2element"];
```

- odwołania do elementów

```
element = mojaTablica[0];
```

- dodawanie elementów

```
mojaTablica[3]="czwarty";
```

Metody i własności Array

- `length` rozmiar tablicy,
- `concat(tab1, tab2 ...)` łączy tablice, tworzy nowy obiekt
- `join(separator)` łączy elementy w łańcuch znaków, domyślny `separator: ','`;
- `pop()` usuwa ostatni element
- `push(elem1, elem2 ...)` dodaje elementy
- `sort(fporównująca)` sortuje tablice, funkcja porównująca w postaci: `funkcja(a, b)`; zwraca `-1` gdy `a < b`, `0` gdy `a == b` oraz `1` gdy `a > b`.

Tworzenie własnych obiektów

- konstruktor

```
function samochod(marka, model, rok) {  
    this.marka=marka;  
    this.model=model;  
    this.rok=rok; }  
}
```

- tworzenie obiektu

```
mojSamochod=new samochod("Ford", "Crown Victoria", 2000  
objectName = {property_1:value_1, property_2:value_2,  
    ..., property_n:value_n};
```

- własności typu obiektu (klasy)

```
samochod.prototype.kolor=null;  
mojSamochod.kolor="bialy";
```

- usuwanie obiektu

```
delete mojSamochod;  
delete mojSamochod.rok;
```

Tworzenie własnych obiektów cd.

- przyporządkowanie metod

```
obiekt.nazwaMetody = nazwaFunkcji  
obiekt.nazwaMetody = function(param1,param2) {...}
```

- wywołanie metody

```
obiekt.nazwaMetody(arg1, arg2 ...);
```

- przykład

```
function wyswietlSam(){  
    var napis = " Piękny " + this.marka " " + this.model;  
    wypisz(napis);  
}  
function samochod(marka, model, rok){  
    this.marka=marka;  
    this.model=model;  
    this.rok=rok;  
    this.wyswietlSam = wyswietlSam;  
}
```

Obiekt typu Math

własność	opis
PI	liczba π
E	podstawa log. naturalnego
metoda	opis
abs(x)	wartość bezwzględna
sin(x), cos(x), tan(x)	f.trygonometryczne
sqrt(x)	pierwiastek kwadratowy
pow(x, y)	x^y
round(x)	zaokrąglenie
ceil(x)	zaokrąglenie w górę
floor(x)	zaokrąglenie w dół
random	liczba pseudolosowa $\in [0, 1]$

Nie należy tworzyć obiektów typu Math!

Obiekt typu `String`

Należy rozróżnić pomiędzy obiektem typu `String` a zmienną typu “łańcuch znaków”

```
s1 = "mój łańcuch znaków";  
s2 = new String("mój obiekt String");
```

W praktyce można używać metod oraz własności obiektu typu `String` na zmiennej typu “łańcuch znaków”. W takim przypadku *JavaScript* dokona tymczasowej konwersji łańcucha znaków do obiektu typu `String`.

Hierarchia typów obiektów

FIXME

I. Wojnicki, JiTW

Spis Treści

- 1 Wstęp
- 2 Elementy Języka
- 3 Sterowanie
- 4 Funkcje
- 5 Obiekty
- 6 HTML**
- 7 Przeglądarka

I. Wojnicki, JiTW

Osadzanie *JavaScript* w HTML

Historycznie - niezgodne z HTML 4.01 Strict!!!
Znacznik końca elementu jest *wymagany*.

```
<html ...>
<head>
<script language="Javascript1.3">
<!-- aby ukryć kod JavaScript
....
// -->
</script>
</head>
<body>
<script language="Javascript1.3">
<!-- aby ukryć kod JavaScript
....
// -->
</script>
</body>
</html>
```

Osadzanie *JavaScript* w HTML

```
<meta http-equiv="Content-Script-Type"  
      content="text/javascript" />  
  
<script type="text/javascript">  
<!-- aby ukryć kod JavaScript  
  
/* kod JavaScript */  
  
// -->  
</script>
```

Osadzanie *JavaScript* w HTML cd.

- podawanie URL pliku z kodem
`<script src="mojeskrypty.js" > </script>`
- dla przeglądarek bez *JavaScript*
`<noscript> ... </noscript>`

Spis Treści

- 1 Wstęp
- 2 Elementy Języka
- 3 Sterowanie
- 4 Funkcje
- 5 Obiekty
- 6 HTML
- 7 Przeglądarka**

I. Wojnicki, JiTW

Obsługa zdarzeń

- ogólna postać

```
<znacznik zdarzenie=" kod JavaScript ">
```

- przykład

```
<input type="button"  
value="oblicz"  
onClick="oblicz(this.form)">
```

`this` dotyczy obiektu, który wygenerował zdarzenie, w tym przypadku jest to `button`; `this.form` dotyczy formularza, w którym umieszczony jest przycisk.

Zdarzenia

Dla wersji JavaScript 1.1 i wcześniejszych nazwy zdarzeń muszą być pisane małymi literami.

zdarzenie	opis
onClick	naciśnięcie przycisku będącego elementem formularza
onChange	zmiana wartości elementu (np. pola tekstowego)
onKeyPress	naciśnięcie klawisza
onMouseOut	kursor myszy opuszcza obiekt
onMouseOver	kursor myszy nad obiektem
onReset	kasowanie zawartości formularza
onSubmit	wysyłanie formularza

Obsługa zdarzeń

```
<h1 onclick="alert('ooooo co tam?')">....</h1>
```

Lepiej użyć `void` (inaczej różne zachowania różnych przeglądarek):

```
<a href="javascript:void(f())">....</a>
```

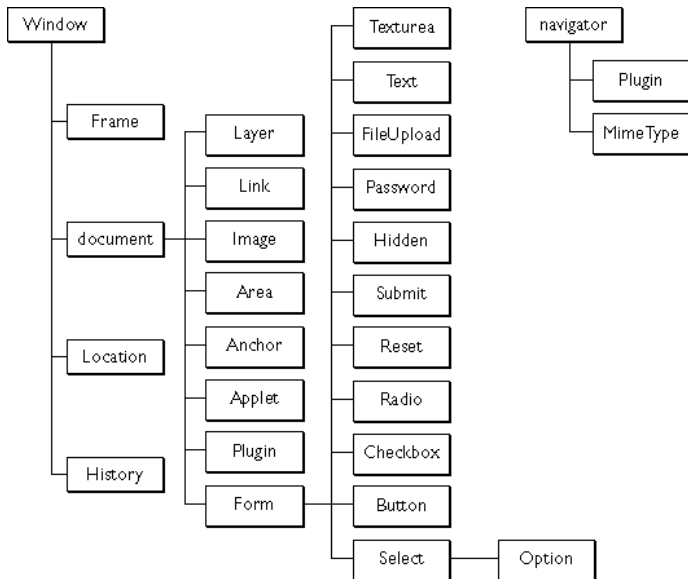
Jeżeli `g()` zwróci `true`, przeglądarka przejdzie pod wskazany adres (podobnie `onSubmit`, `onReset`).

```
<a href="http://w.d.r/" onclick="return g()">....</a>
```

Najpierw wykonywana jest funkcja zdarzenia:

```
<a href="javascript:void(f())" onclick="g()">....</a>
```

DOM: Document Object Model



DOM cd.

Każdy dokument (x)html zawiera następujące obiekty:

- `navigator`: posiada własności zawierające nazwę i wersję przeglądarki, zainstalowane *Plug-Ins*, oraz *MIME*
- `window`: okno
- `document`: własności związane z dokumentem: tytuł, kolor, linki, formularze, `document.write("Hello World!")`
- `location`: własności związane z aktualnym URI,
- `history`: historia

DOM cd.

Każdy obiekt HTML posiada swój odpowiednik DOM dostępny z poziomu *JavaScript* np.

```
document.mojform.tekst1.value
```

jest referencją do wartości pola tekstowego `tekst1` w formularzu `mojform` (wartość atr. `name`) aktualnego dokumentu.

```
document.forms[0].elements[0].value
```

```
document.forms[0].action="...."
```

```
document.forms[0].submit()
```

```
document.forms[0].reset()
```

Obiekty: window

- `open()`, `close()`: otwieranie i zamykanie okien przeglądarki,
- `alert()` wyświetla komunikat w oknie dialogowym,
- `confirm()` okno dialogowe Akceptuj/Porzuć,
- `prompt()` okno dialogowe z polem tekstowym,
- `scrollTo()` przewija zawartość okna,
- `setTimeout()` wywołuje funkcję po upłygnięciu zadanego czasu,
- `location` aktualny URL,
- `status` komunikat w pasku komunikatów.

Obiekt `window` `cd`.

- otwieranie nowego okna,

```
okno=window.open(url,nazwa,opcje);
```

przykładowo:

```
mojeOkno=window.open("http://www.agh.edu.pl")
```

- zamykanie bieżącego okna,

```
window.close() albo self.close(),
```

- zamykanie wskazanego okna,

```
mojeOkno.close().
```

Obsługa formularzy

```
<script>
function ff(){return 'funkcyjka';}
moja=ff();
document.write(moja);
function sub(){return confirm('Potwierdz');}
</script>
<form name="formul1" onsubmit="return sub()">
  <input type="text" name="t1">
  <input type="submit"> </form>
<form name="formul2">
  <input type="text" name="t2">
  <input type="submit" onclick="return sub()"></form>
<script>
  document.formul1.t1.value=ff();
</script>
```


Spis Treści I

```
window.onload = createTOC;
function createTOC() {
    // find the nodes to be added to the Page TOC
    var tocTargets = new Array()
    nodes = document.body.childNodes
    for (var i = 0; i < nodes.length; i++) {
        nn = nodes[i].nodeName
        if (nn == "H1" || nn == "H2" || nn == "H3") {
            tocTargets.push(nodes[i])
        }
    }
    tocDiv = document.getElementById('pageToc')
    // Remove toc if none or one heading
    if (tocTargets.length <= 1) {
        tocDiv.parentNode.removeChild(tocDiv)
        return;
    }
}
```

Spis Treści II

```
}  
// Add the toc contents  
tocDiv = document.createElement('div')  
tocDiv.id = 'pageToc'  
document.body.insertBefore(tocDiv, document.body.firstChild)  
tocList = document.createElement('ul')  
tocList.className = "pageToc"  
tocDiv.appendChild(tocList)  
// Insert elements into our table of contents  
for (var i = 0; i < tocTargets.length; i++) {  
    tocTarget = tocTargets[i]  
    if (tocTarget.id == '') {  
        tocTarget.id = 'pageToc' + i  
    }  
    newItem = document.createElement('li')  
    newItem.className = "pageToc" + tocTarget.nodeName  
    newLink = document.createElement('a')
```

Spis Treści III

```
newLink.href = '#' + tocTarget.id
newLink.innerHTML = tocTarget.innerHTML
newItem.appendChild(newLink)
tocList.appendChild(newItem)
newItem.innerHTML = newItem.innerHTML + ' '
}
}
```

I. Wojnicki, JEW

Obsługa formularza I

Wymagane pola formularza: DATA_*

Wybrane produkty: PROD_*

submit onClick=SendTest

button onClick=ClearProd

```
function SendTest(frm) {
    bad=0;
    badfields="";
    for (var i=0; i < frm.elements.length; ++i) {
        // Get the current field
        form_field = frm.elements[i];
        // Get the field's name
        form_name = form_field.name;
        if (form_name.substring(0,4) == "DATA") {
            sname=form_name.substring(5,form_name.length);
            if (form_field.value==""){
                badfields=badfields+"\n"+sname;
            }
        }
    }
}
```

Obsługa formularza II

```
        bad=1;
    }
}
if (bad) {
    alert("Wypełnij pola:\n" + badfields);
    return false;
}
return confirm('Czy chcesz wysłać zamówienie?');
}
```

Obsługa formularza III

```
function ClearProd(frm) {  
    // Run through all the form fields  
    for (var i=0; i < frm.elements.length; ++i) {  
        // Get the current field  
        form_field = frm.elements[i];  
        // Get the field's name  
        form_name = form_field.name;  
        // Is it a "product" field?  
        if (form_name.substring(0,4) == "PROD") {  
            form_field.value="";  
        }  
    }  
    frm.TOTAL.value="";  
}
```

Cookies

- Dostęp do mechanizmu *cookie* za pomocą: `document.cookie`
- Mechanizm pozwalający przechowywać informacje po stronie przeglądarki w postaci:
`nazwa=wartość; expires=data; domain=domena; path=ściezka`
- Jeżeli `nazwa` bądź `wartość` zawierają średnik, przecinek lub spację należy użyci funkcji `escape(1an)` do zakodowania nazw i odpowiednio `unescape(1an)` do odkodowania.
- `data` musi być w formacie *GMT*, konwersja za pomocą metody `toGMTString()` obiektu `Date`:
`Thu, 2 Aug 2001 20:47:11 UTC.`

Firebug

- JavaScript Debugger
- Profiler
- Edytor HTML/CSS
- Rozszerzenie Firefox'a
- <http://www.getfirebug.com>

Inspect | **h1.siteTitle** < div.contentInner < div.contentInner3 < div.contentInner2 < di

Console | **HTML** | CSS | Script | DOM | Net | Options ▾

```

<div class="contentTop1">
  <div class="contentInner1">
    <div class="contentInner2">
      <div class="contentInner3">
        <div class="contentInner">
          <h1 class="siteTitle">
            <a href="/" style="">
          </h1>
          <div class="subTitle">Thoughts
            on software and life.</div>
          <div class="pageHead">
          <div id="post000170" class="blogPost">
          <div id="post000169" class="blogPost">

```

Style | Layout | DOM | Options ▾

```

.siteTitle {
  font-size: 1.5em;
}
h1 {
  font-size: 1.3em;
  letter-spacing: -1px;
}
Inherited from div#content
#content {
  text-align: left;
}
Inherited from body

```

Done

AJAX

AJAX = Asynchronous JavaScript and XML.

I. Wojnicki JitW

AJAX, Przykład I

```
<p>Server time is: <strong id="stime">Please wait...</strong>
<p>Client time is: <strong id="ctime">Please wait...</strong>
<script type="text/javascript">
function updateTime() {
    date = new Date();
    element = document.getElementById('ctime');
    element.innerHTML = date.getHours() + ':' +
                        date.getMinutes() + ':' +
                        date.getSeconds();
    window.setTimeout('updateTime()', 1000);
}

function updateAjax() {
    xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState==4 && xmlhttp.status==200) {
```

AJAX, Przykład II

```
        document.getElementById("stime").innerHTML =
            xmlhttp.responseText;
    }
}
xmlhttp.open("GET", "date.php", true);
xmlhttp.send();
window.setTimeout("updateAjax()", 1000);
}

window.setTimeout("updateTime(); updateAjax();", 5000);
</script>
```

XMLHttpRequest

- `open (metoda, url, async)` – metoda: **GET/ POST**; `async`: **true/false**
- `send (string)` – **opcjonalny łańcuch znaków dla POST**
- `setRequestHeader ("Content-type", "...")`;
- `onreadystatechange` – **funkcja uruchamiana dla danych asynchronicznych**
- `readyState` – **stan obiektu**
- `status` – **status odpowiedzi serwera**
- `responseText` – **odpowiedź serwera jako tekst**
- `responseXML` – **odpowiedź serwer jako XML (możliwość przetwarzania XML w JavaScript!!!)**

readyState i status

readyState

- 0 – unsent
- 1 – opened
- 2 – headers received
- 3 – loading
- 4 – done

status

- jak dla HTTP (200 – OK, etc.)

AJAX, server push

FIXME: połączenie otwarte aż serwer coś przysła UWAGA:
zalecenie nie więcej niż 2 połączenia równoległe z tym samym
serwerem po stronie klienta

I. Wojnicki, JitW

AJAX, podtrzymanie połączenia I

```
<p>server time is: <strong id="stime">Please wait...</strong>
```

```
function updateAjax() {  
  xmlhttp = new XMLHttpRequest();  
  xmlhttp.onreadystatechange = function() {  
    if (xmlhttp.readyState==3 && xmlhttp.status==200) {  
      document.getElementById("stime").innerHTML=  
        xmlhttp.responseText;  
    }  
    if (xmlhttp.readyState==4) { // server closing conn.  
      xmlhttp.open("GET", "date-sleep.php", true);  
      xmlhttp.send();  
    }  
  }  
  xmlhttp.open("GET", "date-sleep.php", true); xmlhttp.send();  
}  
window.setTimeout("updateAjax();", 1000);
```

AJAX, podtrzymanie połączenia II

```
<?php
  for ($i=0; $i<5; $i++) {
    echo date("H:i:s ");
    ob_flush(); flush();
    sleep(2);
  }
?>
```

http://en.wikipedia.org/wiki/Comet_%28programming%29